Thesis for the degree of Licentiate of Engineering

On Prism-based Motion Blur and Locking-proof Tetrahedra

Mads J. L. Rønnow

Division of Interaction Design
Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2021

On Prism-based Motion Blur and Locking-proof Tetrahedra
Mads J. L. Rønnow

ISSN 1652-876X
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Göteborg, Sweden
Phone: +46(0)32 772 1000

Printed in Sweden
Chalmers Reproservice
Göteborg, Sweden 2021

On Prism-based Motion Blur and Locking-proof Tetrahedra

Mads J. L. Rønnow
Department of Computer Science and Engineering
Chalmers University of Technology

Thesis for the degree of Licentiate of Engineering
a Swedish degree between M.Sc. and Ph.D.

Abstract

Motion blur is an important visual effect in computer graphics for both real-time, interactive, and offline applications. Current methods offer either slow and accurate solutions for offline ray tracing applications, or fast and inaccurate solutions for real-time applications.

This thesis is a collection of three papers, two of which address the need for motion blur solutions that cater to applications that need to be accurate and as well as interactive, and a third that addresses the problem of locking in standard FEM simulations. In short, this thesis deals with the problem of representing continuous motion in a discrete setting.

In **Paper I**, we implement a GPU based fast analytical motion blur renderer. Using ray/triangular prism intersections to determine triangle visibility and shading, we achieve interactive frame rates.

In **Paper II**, we show and address the limitations of using prisms as approximations of the triangle swept volume. A hybrid method of prism intersections and time-dependent edge equations is used to overcome the limitations of Paper I.

In **Paper III**, we provide a solution that alleviates volumetric locking in standard Neo-Hookean FEM simulations without resorting to higher order interpolation.

Keywords: motion blur, real time, parallel, GPU, prism, continuous collision detection, locking, FEM

Acknowledgements

I want to thank my supervisor Marco Fratarcangeli for accepting me as a PhD student and going above and beyond to support me in my research and for his everlasting patience and understanding. I also want to thank my co-supervisor Ulf Assarsson for his significant involvement and for spending his precious time collaborating to solve the hard problems.

I want to thank my colleagues Tomasz Kosinski, Yuchong Zhang, and Ziming Wang for their friendship and for making each day more fun and enjoyable.

Finally, I want to thank my partner Cindie for her love and support. It is such a privilege to have you in my life and I am deeply thankful that I am able to share it with you.

List of Appended Papers

This thesis is a summary of three papers.
References to the papers will be made with roman numerals.

Paper I - Mads J. L. Rønnow, Ulf Assarsson, Marco Fratarcangeli,
Fast Analytical Motion Blur with Transparency,
Computers & Graphics, 95, April 2021, 36-46

Paper II - Mads J. L. Rønnow, Ulf Assarsson, Marco Fratarcangeli,
Improved Accuracy for Prism-based Motion Blur,
Submission Pending

Paper III - Mihai Frâncu, Arni Asgeirsson, Mads J. L. Rønnow, Kenny Erleben,
Locking-Proof Tetrahedra,
ACM Transactions on Graphics,
Accepted, Pending Publication

Table of Contents

I Summary

Part I
Summary

1 Introduction

One of the main goals of Computer Graphics is to visualize virtual scenes as truthful to our own perception of reality as possible, including the perception of real world cameras used for recording still images or video.

A camera, produces an image by having the shutter open for some short amount of time in order to collect sufficient light to produce a clear image. In essence, a photo taken from a camera is not a single instance of time, but a range. If there is any movement within this range, it produces a blurring effect in the photo.

In Computer Graphics, the goal is thus to produce an image similar to what a human eye or a camera would produce, including the blur effect that happens as a result of movement within the aforementioned range. Due to the discrete nature of the computer animation, we have to produce this effect with just two instances of time instead of a range, i.e. the time of opening the shutter and the time of closing the shutter, that is defined as the *exposure time*.

Modern visual effects applications involve interaction that demands real-time visual feedback. Therefore, there is a continuous strive to compute visual effects such as motion blur at interactive frame rates. For these applications, we must aim to ensure that the computation time is low enough to enable interactive frames rates while at the same time maintaining a high visual fidelity.

1.1 Motion blur

Fig. 1 shows an example of a scene with and without motion blur. The motion blur adds an impression of motion in a still image that would otherwise appear frozen in time. The effect is especially important for motion pictures that are typically filmed at 24 frames per second. At these relatively low frame rates, the motion between frames can be quite large. Hence the blur is also quite pronounced, and it is important to reproduce accurately.

A moving object in an image appears transparent and blurred. The degree of transparency and blur depend on the exposure time and the relative displacement of the object on the image.

The degree of transparency of an object in the image depends on how long it covers the same area of the image within the exposure time. This is complicated by an object's surface having non-uniform colors and characteristics. An opaque object that covers a certain area of the image during the entire exposure time would not be transparent at all in that area. If the object would cover the same area just for a fraction of the exposure time,

Figure 1: The same scene without motion blur (left) and with motion blur (right). The motion blur adds an impression of movement in the still image.

then it would appear partially transparent. How blurred an area of an object is depends on how far a portion of the object moves in image space within the exposure time.

1.2 Discrete setting

The physical world can generally be regarded as continuous both in space and time. Computers, however, use numerical methods that usually are discrete. Time has to be considered at distinct "snapshots" and progresses in discrete steps. Space is represented in discrete sizes and distances, usually with triangles, and ultimately as discrete pixels on the screen.

Time is typically discretized in steps and by assuming linear or a certain degree of non-linear motion between steps. For several applications, however, the continuous information is still needed. Motion blur and collision detection are two such applications. Motion blur is one such case because it occurs as a result of the continuous motion. In the case of collision detection, the positions of objects between the discrete points in time are equally important to check for collision as not to miss collisions that happen between time steps. Also, collisions are usually instantaneous phenomena happening in-between a time step. So, they may be lost if the state of the animation is updated only at the end of the time step.

In a discrete animation we only have single positions in time and the object's trajectory, not the continuous motion. How do we accurately reproduce the blurry motion in between? And how do we do so without it being prohibitively computationally expensive?

2 Previous approaches

Achieving the motion blur effect effectively and accurately presents itself as a difficult problem in computer graphics.

A simple brute-force method for achieving the motion blur effect is to render the scene as usual but in hundreds or thousands of tiny steps and accumulate all the images into a single aggregate that represents an average image, see Fig. 2.

Figure 2: Results from brute-force method with five iterations (left), ten iterations (middle), and 100 iterations (right). With five and ten iterations the individual frames are still visible, while for 100 they have become indistinguishable.

This, however, is intractable for most applications due to the need for rendering the scenes hundreds or thousands of times per time step. In a way, this naive method tries to make the discreteness invisible by taking very small steps. While the brute-force method is slow it produces good results and can be regarded as ground truth.

We aim for a method that does not need to take many intermediate steps, but instead fills in the intermediate information based on the start and end configurations of the time step. Our focus is on real-time applications, and a solution that fits into existing graphics pipelines optimized for rasterizing triangles.

2.1 Post-process approaches

Many methods used for real-time graphics applications use a post-process technique [7, 8, 11] to achieve the motion blur effect. In these methods,

motion vectors are recorded in the frame buffer and the motion blur is computed in screen-space. This achieves good results at very low cost, but there are cases, that are not handled well by screen-space methods, such as transparency and when objects in the same area of the screen move in different directions. One such case is pictured in Fig. 3.

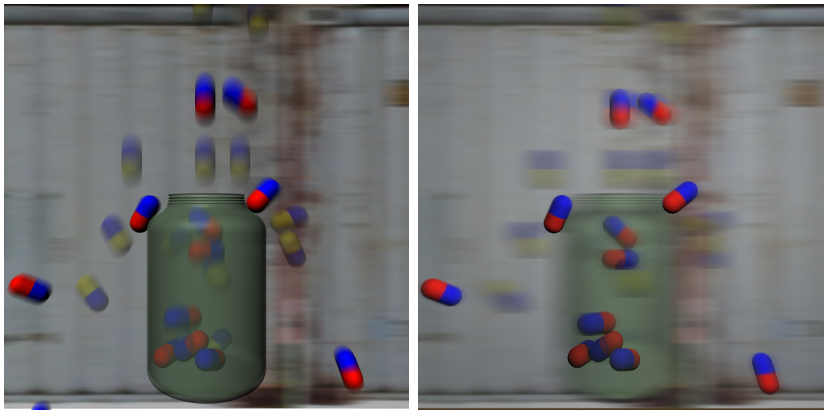

Figure 3: Pills falling into a bottle in front of a horizontally moving background. Rendered with brute-force ground-truth method (left) and a post-process method (right). The post-process method produces incorrect blur on the semi-transparent bottle and pills, and the opaque pills are blurred largely in the direction of motion of the background instead of their own.

2.2 Stochastic approaches

Other methods are based on stochastic sampling [1, 2, 3, 4, 5, 12, 15]. Here the triangle's motion is bounded in screen-space, and random samples are taken in the triangle's area of motion. These methods are typically faster than the naive brute force method mentioned earlier and they can produce the same precise results at high sample rates, but they are still not fast enough for real-time applications without lowering the sample rate and introducing visual artifacts. These methods are typically used for offline ray tracing applications, where image quality is a priority and real-time performance is less of a concern.

2.3 Bridging the discrete - continuous gap

If we assume that all motion between the two points in time is linear, then we can linearly interpolate between the two configurations and "fill in the gaps" analytically.

2.3.1 Analytical motion blur

This approach works similarly to stochastic methods but the color contributions are computed analytically to get a clean and precise image [5, 6, 9].

As a triangle moves over the screen and passes through pixels, the duration of time a triangle is present in the pixel determines the opacity of the triangle in said pixel. While the color of the pixel is determined by the colors on the area of the triangle that passes through the pixel. In order words, if we can determine how long a triangle is present in a pixel and which area of the triangle covers the pixel, then we can compute the moving triangle's contribution to the pixel's color. To account for multiple triangles covering the same pixel within a time step, depth order and time collision also needs to be taken into consideration. For this reason, we also need to keep track of what specific time range the triangle covers a pixel and at what depth.

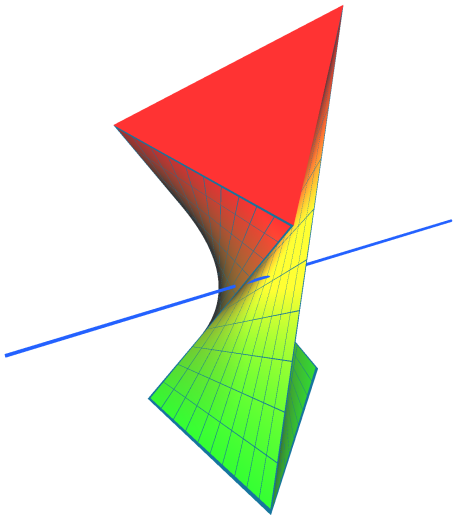

Figure 4: Swept volume of moving triangle taking the shape of a triangular prism. The green to red color gradient embedded on the prism surface represents the progression of time from the beginning to the end of the time step. The blue line segment's intersections with the prism surface indicate all the entry and exit points of the triangle's presence in a pixel.

With this in mind, the triangle's entry and exit time in a pixel can be computed in a number of ways. The surface of the moving triangle is computed, either by time dependent edge equations [5] or as the surface of a triangular prism [9, 14], which bounds the swept volume that the triangle

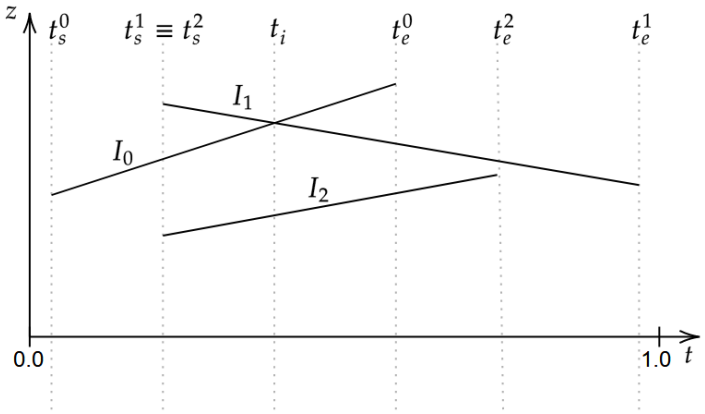

Figure 5: Intervals that overlap in time need to partitioned such that their parts can be handled in the right depth order. Interval I_0 is in front of interval I_1 from time t_s^0 up until time t_i, where the depth order of the intervals is swapped.

traverses. These entry and exit points are known as intervals and typically also include information regarding entry and exit depth, normals, and UV values. Fig. 4 depicts the swept volume of a moving triangle, with the color indicating the time from 0 (red) to 1 (green). The blue line represents the triangle's entry and exit points in a pixel, shown as line segment intersections with the prism from the point of view of a given pixel.

Resolving time collisions. Triangles that are present in the same pixel but at different times within the exposure time do not occlude each other, and simply contribute the same weight to the final color of the pixel. If the triangles are present in the same pixel at overlapping times, it is equivalent to if the triangles were non-moving and occluding each other - they need to be processed in the correct depth order, and in the case of motion blur rendering, only for the duration of time they overlap. Fig. 5 shows an example with several intervals that partially overlap. In order to ensure correct depth order, the intervals must be partitioned according to the overlap.

Alpha blended transparency is handled almost implicitly with this method, because the intervals with time overlap need to be sorted by depth.

2.3.2 Prisms

As the moving triangle goes from its starting position to its ending position, it passes through a volume of space which has the shape of a triangular prism.

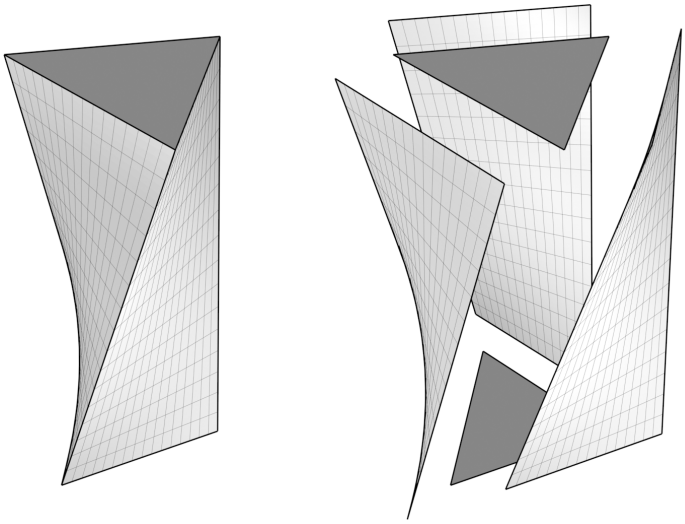

Figure 6: Modelling the swept volume surface as five separate surfaces, three bilinear patches and two triangles.

In general, 3D scenes are composed of objects represented by triangular surface meshes. A scene is rendered as an instance of time. This existing paradigm can be modified to instead render a range of time between two points in time, by modelling the objects as surface meshes where each original triangle is extruded to form a triangular prism as seen on Fig 6.

A given triangle on the surface mesh has a start position and an end position within the shutter range. These two positions are used to form the triangular prism, and the prism sides represent the continuous motion of the triangle edges during the time where the shutter is open, represented by the color gradients on the prism in Fig. 4.

A prism side is a bilinear patch, that is, a 3D quadrilateral with a potentially curved surface.

By retrieving information relating to position in world-space, time, and triangle orientation for every sampled point on this bilinear patch, we can successfully render the triangle's continuous motion between the two points in time.

3 Summary of Included Papers

In this Section, we will describe the main contributions of each paper:

Paper I introduces a novel GPU method for analytical motion blur.

Paper II presents shortcomings and a solution for prism based modelling of linear triangle motion and motion blur.

Paper III describes a method for locking-free simulation of soft body incompressible materials.

3.1 Paper I - Fast Analytical Motion Blur with Transparency

Problem: Parallelizing analytical motion blur for GPU architectures is challenging due to the need to consider depth and time overlap and the order of the rendered moving triangles.

Methodology: A key idea is the use of depth-time intervals to represent the range in which a moving triangle is present in a pixel. For each pixel a list of intervals is generated, accounting for all the triangles that are present in that pixel within the time step. The intervals in the list are first tested for overlap in time, because time overlap means that triangles occlude each other. Intervals that overlap in time must not only be ordered by depth, for correct occlusion culling or alpha blending, but the intervals must also be partitioned for every partial overlap between intervals (see Fig. 7).

Our method builds on the CPU algorithm described by Gribel et al. [5] and is structured in several GPU-based stages as depicted in Fig 8. First the linearly moving triangles are converted into prisms and their clip-space area is bounding by an AABB (1). The AABB's are then rasterized followed by ray intersection tests with all prism surfaces, and the depth complexity of all intersections is established for each pixel (2a), from the depth complexities an exclusive sum is calculated (2b), which is used to determine and allocate the memory to store all intersections in entry-exit pairs (2c), finally the AABB's are rasterized again, this time the intersections are stored in pairs as intervals (2d). Following this, the intervals in each pixel list are sorted by start time to enable simpler time overlap detection (3). The sorted lists are then processed for time overlaps and depth order, resulting in the final pixel colors (4).

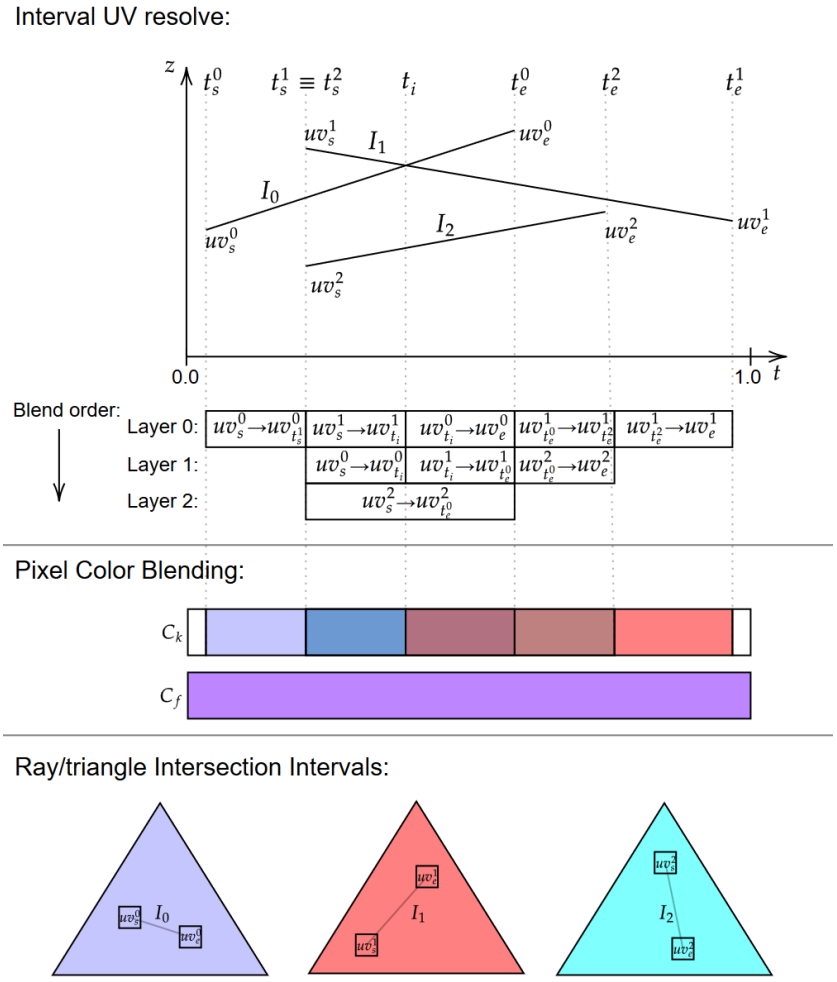

Figure 7: Resolving partially or fully overlapping as well as intersecting intervals.

Contribution: This paper presents a fast GPU method for analytical motion blur that has full support for alpha blended transparency. The paper fills the gap between "fast and inaccurate" and "slow and precise" by taking advantage of the GPU. We are able to obtain real-time rendering (60 FPS at 1080p) of analytical motion blur for the tested scenes.

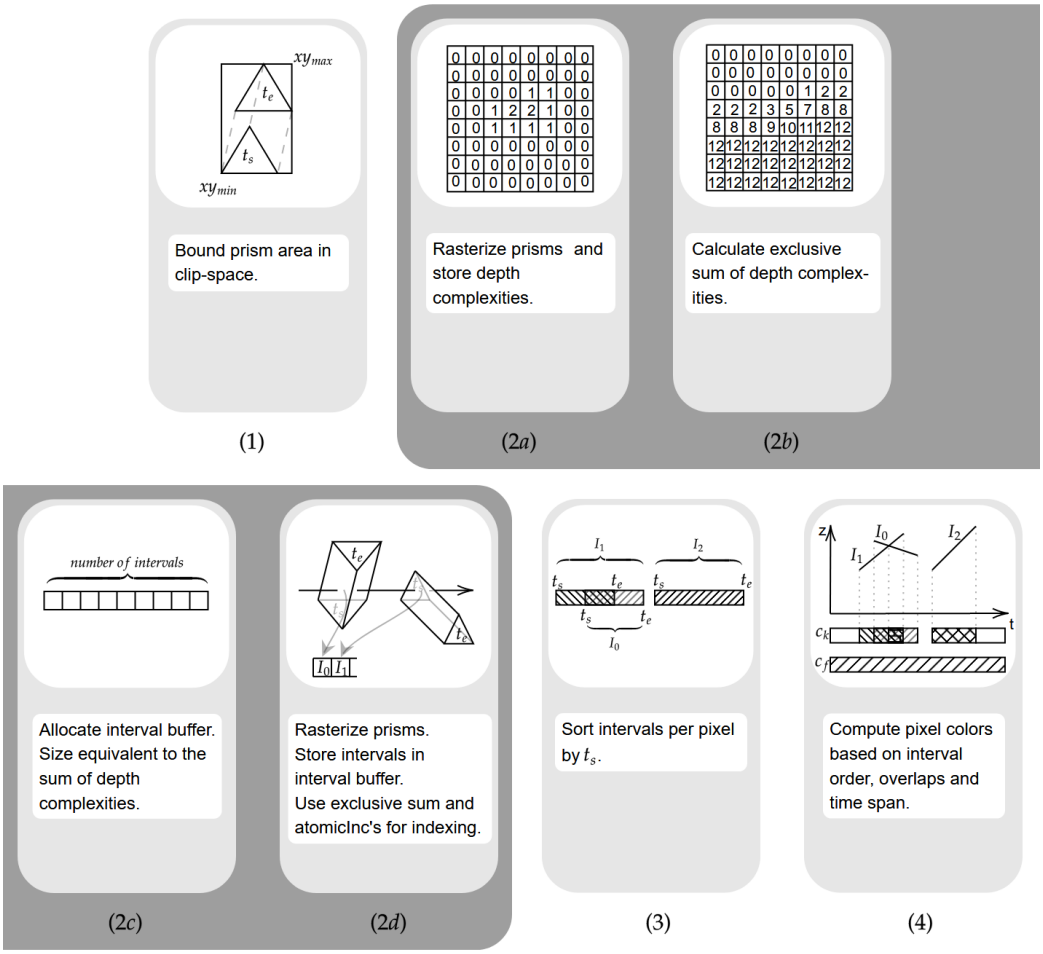

Figure 8: The stages of the motion blur system.

3.2 Paper II - Improved Accuracy for Prism-based Motion Blur

Problem: Prisms are found to be insufficient in bounding the volume swept by linearly moving triangles. This results in incorrect depth, normal, and UV values on the interpolated surfaces of the prisms in cases where the triangle has rotational movement while its vertices move linearly, as shown in Fig 9.

Methodology: Prisms are not enough to capture the correct surface of the swept volume of a moving triangle, but the clip-space area of the prism remains a conservative bound, thus we can still use prisms if we combine them with the use of edge equations to compute precise depth, normal, and UV

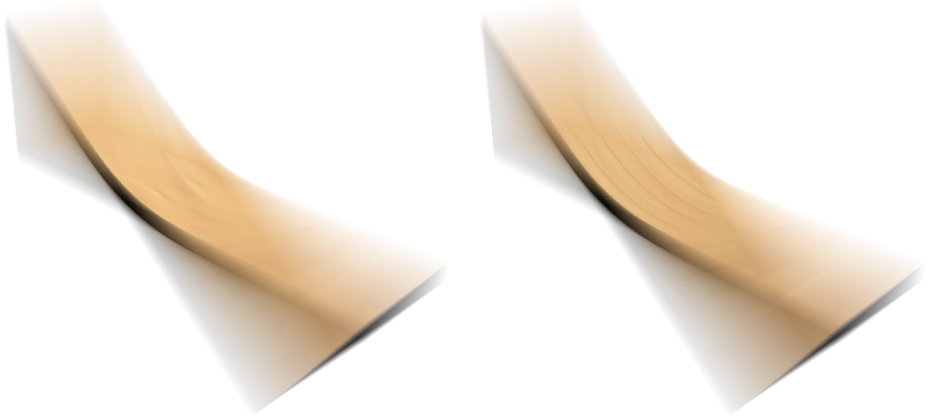

Figure 9: The rotational motion in the middle of the plank causes nonlinearity that is not captured well by the single anistropic lookup (left). With our adaptive subdivision the image quality is markedly improved (right).

values. To ensure correct sampling of the potentially nonlinear interpolated surface we need more samples. We use an adaptive subdivision scheme that depends on how much sampled middle point values deviate from linear interpolation. If the deviation is sufficiently large, another sample point is made at the middle point and the process is repeated recursively. This gives a closer approximation of the true nonlinearity.

Contribution: We have presented theoretically and empirically that prisms fail to conservatively bound the triangle's true swept volume. We have also presented a solution based on hybrid prism/edge equations with adaptive subdivision of intervals based on how much their depth values deviate from being linear, while keeping the performance penalty modest for the improved image quality.

3.3 Paper III - Locking-Proof Tetrahedra

Problem: Incompressible materials in finite element method (FEM) simulations suffer from locking when using coarse tetrahedral meshes. In this work we deal with volumetric locking which manifests due to geometric discretization and usually improves when increasing the mesh resolution and lowering the poisson ratio.

Methodology: The devised method presents a new way to alleviate locking by use of the mixed formulation of nonlinear FEM. The choice of different shape functions allowed by mixed FEM is key to preventing locking. The extra degrees of freedom allow the body to deform and the proper shape function choice can determine whether locking happens or not.

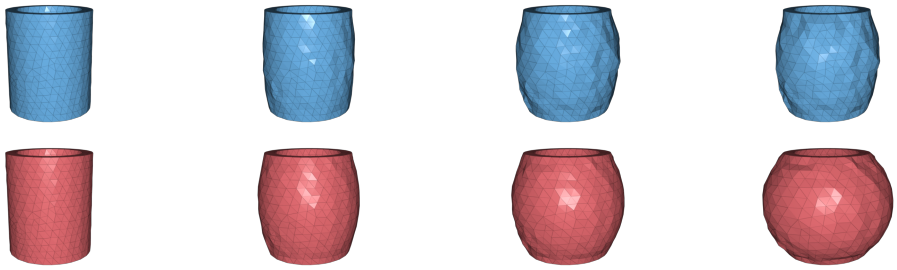

Figure 10: Simulation of an incompressible rubber tube undergoing inflation by internal pressure. The standard Neo-Hookean FEM (top) suffers from locking, while our mixed FEM (bottom) deforms correctly.

Contribution: We describe a novel mixed FEM method that alleviates locking. Our method gives better results and is more robust than standard Neo-Hookean simulations without resorting to higher order interpolation (see Fig. 10).

4 Discussion and Future Work

Our work on motion blur emerged from our investigation of collision detection. The problem of determining when triangles share the same time range was first meant to be applied to collision detection, but we found it has been applied to motion blur in the past, and that the accelerated GPU implementation we had built for continuous collision detection could be adapted for motion blur to improve on this previous work.

In the scope of the future work related to collision detection, during my research I also co-authored the paper on locking-proof tetrahedra using mixed finite element method (FEM) (see appended papers).

4.1 Collision detection

Collision detection involves determining if objects (typically under motion) collide. We restrict our consideration to only triangle based objects. For

a scene with objects made of many thousands or millions of triangles this presents a $O(n^2)$ problem of testing every triangle for intersection with every other triangle.

In order to speed up the process, acceleration structures can be used for spatial partitioning, or the so called: **broad-phase** [10]. This typically involves creating a bounding volume hierarchy (BVH), that facilitates elimination of collision candidates where the bounding objects do not collide.

After the broad phase, we have a list of bounding objects that intersect and we need to check if the triangles inside the bounds are also intersecting - this is known as the **narrow-phase** [10]. Here, triangles are checked for intersection with other triangles. To simplify this intersection testing process one can instead check edges with other edges and triangle faces with vertices [13]. Testing two triangles for intersection with each other then results in 9 edge-edge tests and 6 vertex-face tests. Since triangle edges and vertices are shared in a mesh, however, on average the number of tests per triangle is lower.

Discrete collision detection finds collisions between geometry at a single instance in time. In animating scenes, this can lead to what is known as *tunnelling*, where objects can falsely pass through one another, as depicted in Fig. 11.

In addition, it might also be interesting for us to know the exact time, between two simulation time steps, that the collision takes place, such that the response system can use this exact time.

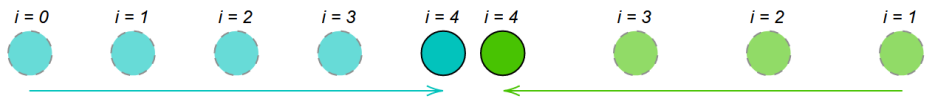

Figure 11: An example of tunneling. The blue and the green ball approach each other in discrete steps. After step 4 the standard collision detection system will miss the collision because the two balls will pass completely through each other, while continuous collision detection would correctly report the collision.

In order to avoid tunnelling and find all collisions between two instances in time we must either shorten the length of time between two time steps in the animation until it can be made certain that no tunnelling can occur, or we can use continuous collision detection.

4.1.1 Continuous collision detection

Continuous collision detection (CCD) aims to find all collisions between two time steps as well as the exact time of the collisions. The typical way to implement continuous collision detection is to add a time dimension to the analytical intersection calculations. The bounding volumes also need to be adjusted so that they contain the entire range of motion of the object within the exposure time. Due to the extra time dimension, continuous collision detection is significantly more expensive than discrete collision detection.

4.2 Multi-view rasterization based continuous collision detection

In our ongoing work on collision detection, we propose to use prisms for continuous collision detection. In order to avoid the complex intersection calculations between time dependent triangles, involving both the processing of broad phase and narrow phase, we can instead apply a ray based approach, where we use this claim:

If a ray consecutively enters two distinct prisms before it exits the first, then these two prisms overlap.

By casting rays and rendering the prisms in the scene from multiple views we can sample the scene for overlapping prisms. If we, as in our motion blur work, also retrieve the information on the prism surfaces, then we can determine not only that the moving triangles overlap the same space within the time step, but also if they collide and the exact time of collision.

Bibliography

[1] Tomas Akenine-Möller, Jacob Munkberg, and Jon Hasselgren. Stochastic rasterization using time-continuous triangles. In *ACM SIGGRAPH/Eurographics Graphics Hardware*, GH '07, page 7–16. Eurographics Association, 2007.

[2] Solomon Boulos, Edward Luong, Kayvon Fatahalian, Henry Moreton, and Pat Hanrahan. Space-time hierarchical occlusion culling for micropolygon rendering with motion blur. In *High Performance Graphics*, page 11–18. Eurographics Association, 2010.

[3] J. S. Brunhaver, K. Fatahalian, and P. Hanrahan. Hardware implementation of micropolygon rasterization with motion and defocus blur. In *High Performance Graphics*, page 1–9. Eurographics Association, 2010.

[4] Kayvon Fatahalian, Edward Luong, Solomon Boulos, Kurt Akeley, William R. Mark, and Pat Hanrahan. Data-parallel rasterization of micropolygons with defocus and motion blur. In *High Performance Graphics*, page 59–68. Eurographics Association, 2009.

[5] Carl Johan Gribel, Michael C Doggett, and Tomas Akenine-Möller. Analytical motion blur rasterization with compression. *High Performance Graphics*, 10, 2010.

[6] Carl Johan Gribel, Jacob Munkberg, Jon Hasselgren, and Tomas Akenine-Möller. Theory and analysis of higher-order motion blur rasterization. In *Proceedings of the 5th High-Performance Graphics Conference*, pages 7–15, 2013.

[7] Jean-Philippe Guertin and Derek Nowrouzezahrai. High Performance Non-linear Motion Blur. In Jaakko Lehtinen and Derek Nowrouzezahrai, editors, *Symposium on Rendering - Experimental Ideas & Implementations*. Eurographics Association, 2015.

[8] Jean-Philippe Guertin, Morgan McGuire, and Derek Nowrouzezahrai. A fast and stable feature-aware motion blur filter. In *High Performance Graphics*, page 51–60. Eurographics Association, 2014.

[9] Minh-Phuoc Hong and Kyoungsu Oh. Real-time motion blur using extruded triangles. *Multimedia Tools Appl.*, 77(11):13323–13341, 2018.

[10] P. M. Hubbard. Collision detection for interactive graphics applications. *IEEE Transactions on Visualization and Computer Graphics*, 1(3):218–230, 1995. doi: 10.1109/2945.466717.

[11] Morgan McGuire, Padraic Hennessy, Michael Bukowski, and Brian Osman. A reconstruction filter for plausible motion blur. In *Interactive 3D Graphics and Games*, page 135–142. ACM, 2012.

[12] Jacob Munkberg and Tomas Akenine-Möller. Backface culling for motion blur and depth of field. *Journal of Graphics Tools*, 15:123–139, 2011.

[13] Xavier Provot. Collision and self-collision handling in cloth model dedicated to design garments. In Daniel Thalmann and Michiel van de Panne, editors, *Computer Animation and Simulation '97*, pages 177–189, Vienna, 1997. Springer Vienna.

[14] Konstantin Shkurko, Cem Yuksel, Daniel Kopta, Ian Mallett, and Erik Brunvand. Time interval ray tracing for motion blur. *IEEE transactions on visualization and computer graphics*, 24(12):3225–3238, 2017.

[15] Karthik Vaidyanathan, Robert Toth, Marco Salvi, Solomon Boulos, and Aaron Lefohn. Adaptive image space shading for motion and defocus blur. In *High-Performance Graphics*, page 13–21. Eurographics Association, 2012.